Technical Report
Linear Regression Models To Predict Cancer Death Rate

CS452 Homework1

Ordinary Least Squares / Gradient Descent / KNN-Regression

ALKIN KORKUT
REYHAN AYDOĞAN
26 November 2023

CONTENTS

1. *Introduction*

Purpose

The primary goal of this work is to create linear regression models to predict cancer death rate from “cancer.csv” file by using ordinary least squares, gradient descent, and knn - regression approaches. In ordinary least squares method, LinearRegression method is used from Scikit-learn library. As for gradient descent approach, it is written a gradient descent algorithm from scratch. Lastly, knn-regression method is used again from Scikit-learn library. Only NumPy, Pandas, Scikit-learn, Matplotlib, Seaborn, and Chardet libraries are used.

Loading and Reading Dataset

First of all, I have encountered an error while trying to read the 'cancer.csv' file. The error is:

“UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in position 41137: invalid continuation byte”

It is about an issue decoding the file as UTF-8. I tried to solve this problem importing 'chardet' library and then tried to open the dataset. 'Chardet' library already installed in the python. Therefore, I did not need to install again.

```python
import chardet
```

```python
# Detect encoding
with open("cancer.csv", 'rb') as f:
    result = chardet.detect(f.read())

# Load the dataset with detected encoding
df = pd.read_csv("cancer.csv", encoding=result['encoding'])
```

2. Data Processing and Visualizing

Preprocessing

Missing data could affect the performance of the models. So, it needs to be handled. Also, we need to remove non-numeric features of the data such as categorical or text data. At the first step, I removed the non-numeric features (#binnedinc, #geography). After purify the data from non-numeric features, I removed the rows with missing data. While performing these operations, I checked the shape of the data frame in the intermediate steps and checked whether there was null data.

Dataset Description

Before the preprocessing, we have 3047 rows and 34 columns at the data frame. Afterwards, we have only 591 rows and 32 columns. Following statistics are available when the code is run.

I. Number of samples.
II. Number of features.
III. Mean and Variance for each feature.
IV. Correlation matrix.

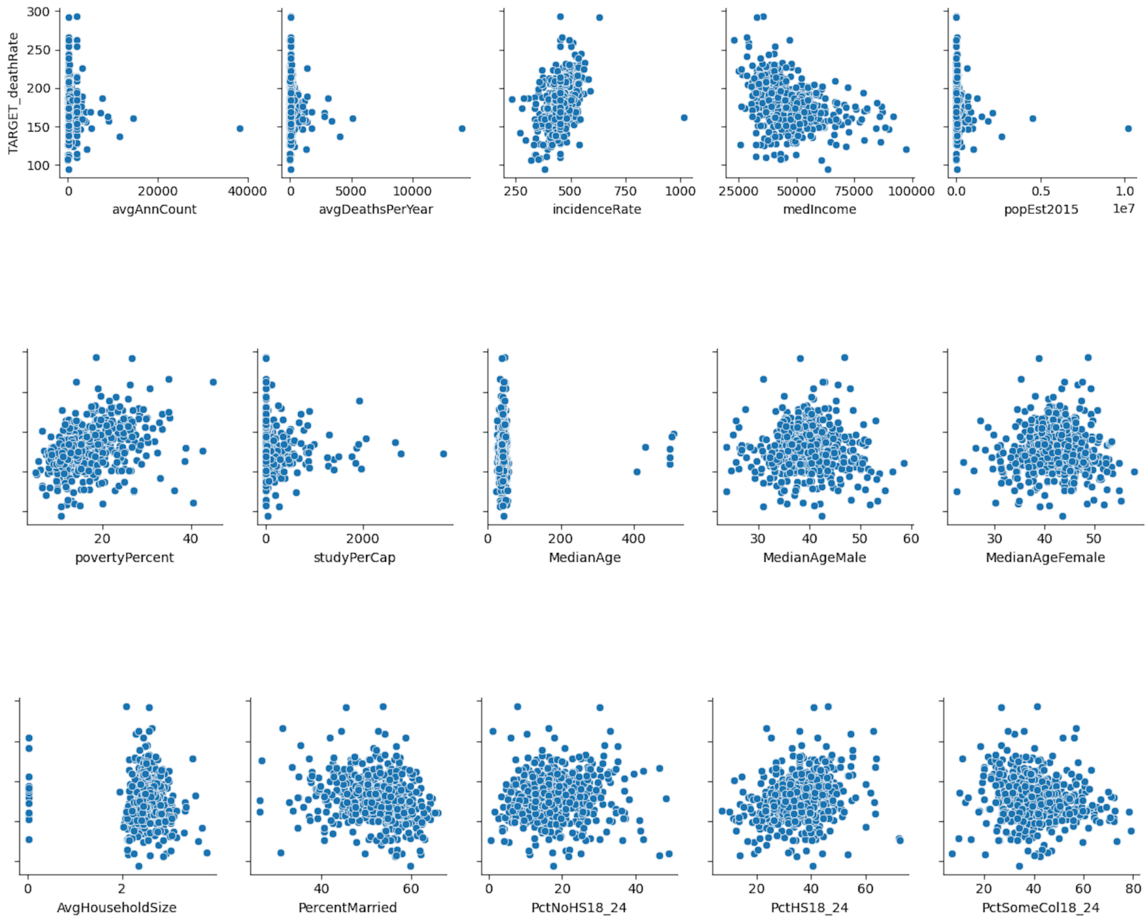
These stats can provide valuable and useful insights for the characteristics of the data. Number of samples includes the total number of instances after preprocessing. It gives an understanding of the dataset's size. Number of features indicates the attributes of the data. It gives an understanding about the dimensionality of the dataset. High variance may indicate significant variability in data points for that feature. When there is low variance, it means that the data points are near the mean. The correlation matrix illustrates the degree of relationship between two features. A high positive correlation between features indicates a tendency for them to rise or fall together. On the other hand, an inverse relationship is suggested by a high negative correlation. Additionally, a low correlation or close to zero indicates a weak or no linear relationship.

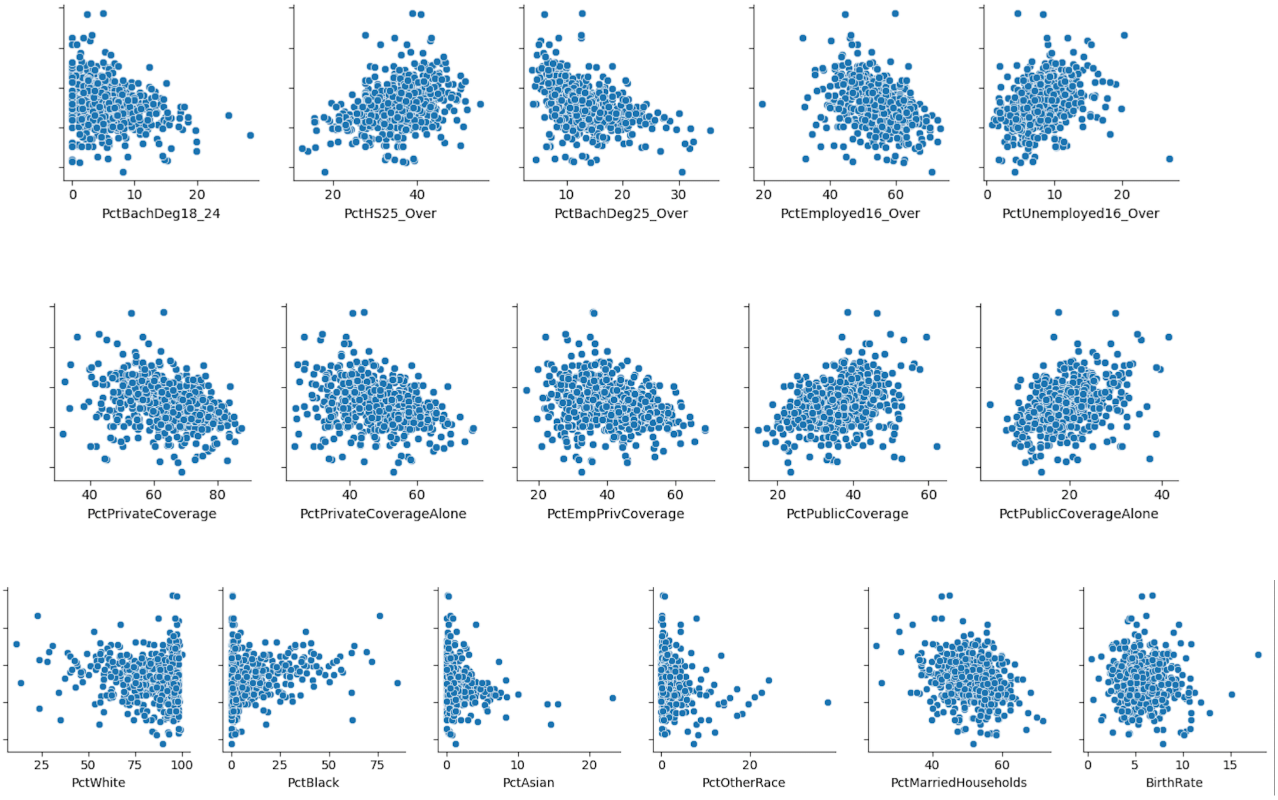
Data Visualization

Before start plotting, first I moved the y_target_variable (TARGET_deathRate) column to the last index of the data frame. After this moving operation I created scatter diagrams to visualize the relationship between the death rate (the target) and other features separately.

```
targetColumn = 'TARGET_deathRate'
# Move the target column to the last index of the dataframe
df = df[[col for col in df.columns if col != targetColumn] + [targetColumn]]

# Create scatter diagrams to visualize the relationship between
# the death rate (the target) and other features separately
sns.pairplot(df, x_vars=df.columns[:-1], y_vars='TARGET_deathRate', kind='scatter')
plt.show()
```

Scatter Diagrams:





Dataset Splitting

Dataset is splitted as test_size = 0.20 and random_state = 271.

Removing Outliers

I removed the outliers using Z-Score technique.

```
# Removing outliers using Z-score technique
z_scores = (X_train - np.mean(X_train, axis=0)) / np.std(X_train, axis=0)
outliers = (np.abs(z_scores) > 3).all(axis=1)
X_train_no_outliers = X_train[~outliers]
y_train_no_outliers = y_train[~outliers]
```

Using this procedure I tried to provide the dataset free from the impact of extreme values. It's a widely used method to ensure that statistical measures fairly represent the vast majority of the data and to clean up data.

Scaling

```
from sklearn.preprocessing import StandardScaler

# Scaling using StandartScaler
scaler = StandardScaler()
X_trainScaled = scaler.fit_transform(X_train_no_outliers)
X_testScaled = scaler.transform(X_test)
```

The aim here is to bring the features onto the same scale. The performance and convergence speed of machine learning algorithms can be enhanced by features that are nearly normally distributed and/or on similar scales. By using standization technique the feature columns are centered at at mean 0 with standard deviation 1 so that the feature columns take the form of a normal distribution.

3. Ordinary Least Squares

Implementation

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

def ordinary_least_squares(X_train, X_test, y_train, y_test, numFeatures):
    # Select the first 'num_features' features.
    selectedFeatures = X_train[:, :numFeatures]

    # Fit the model with LinearRegression().
    model = LinearRegression()
    model.fit(selectedFeatures, y_train)

    # Make predictions on the test set.
    selectedFeaturesTest = X_test[:, :numFeatures]
    y_pred = model.predict(selectedFeaturesTest)

    # Evaluate the model by calculating mse, mae, and r2 score.
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    return model, mse, mae, r2

results = {}

# Evaluating models for F = 5, 10, 20 values
for numOfFeatures in [5, 10, 20]:
    model, mse, mae, r2 = ordinary_least_squares(X_trainScaled, X_testScaled, y_train, y_test, numOfFeatures)

    # Storing results in the dictionary
    results[numOfFeatures] = {
        'f_value': numOfFeatures,
        'model': model,
        'mse': mse,
        'mae': mae,
        'r2': r2
    }

    print(f"\nResults for {numOfFeatures} features:")
    print(f"MSE: {mse}")
    print(f"MAE: {mae}")
    print(f"R2 Score: {r2}")
    print(f"Coefficients: {model.coef_}")
    print(f"Intercept: {model.intercept_}")


# Identifying the best model (The best model is with lowest MSE)
bestModelKey = min(results, key=lambda k: results[k]['mse'])
bestModel = results[bestModelKey]['model']

print("\n")
print("---- RESULT ----")
# Printing the best model
print("The Best Model with f value", results[bestModelKey]['f_value'])
print(f"{bestModel} with mse:", results[bestModelKey]['mse'])
print("with mae:", results[bestModelKey]['mae'])
print("with R2 score:", results[bestModelKey]['r2'])


# Printing the linear equation for the best model
print("\nLinear Equation for the Best Model:")
equation = f"y = {bestModel.intercept_} "
for i, coef in enumerate(bestModel.coef_):
    equation += f"+ ({coef} * X_{i}) "
print(equation)
```

Model Evaluation

Results for 5 features:

MSE: 463.51974876777234

MAE: 16.07056016183814

R2 Score: 0.3100302483143771

Coefficients: [-4.79333086 10.0087527 10.12404938 -11.27436885 -5.35973882]

Intercept: 179.33326271186442

Results for 10 features:

MSE: 443.1639611822952

MAE: 15.64879809384971

R2 Score: 0.34033074304637934

Coefficients: [-4.37843822 8.99460824 10.25193451 -7.60562729 -5.0248862 4.01793302 -0.44724573 -0.82089988 -2.19776472 3.79246302]

Intercept: 179.33326271186442

Results for 20 features:

MSE: 446.0138685405535

MAE: 15.308115424103127

R2 Score: 0.3360885292517528

Coefficients: [-1.79565472e+00 7.15418346e+00 9.31413288e+00 -9.13674545e-01 -4.42582484e+00 2.14458324e+00 3.57875570e-02 -1.22942575e+00 -4.95286280e-01 -3.42388629e-02 -3.28715947e-01 -7.88766000e-01 -7.28401451e+01 -8.03662482e+01 -9.88841428e+01 -4.05484774e+01 4.49185785e+00 -4.33834617e+00 -1.39613724e+00 3.20616655e+00]

Intercept: 179.33326271186442

MSE measures the average squared difference between actual and predict values. When we look at the MSE results, lower mean-squared-error indicates better model performance. But we should know it MSE is sensitive to outliers since it squares the differences. On the other hand mean-absolute-error is less sensitive to outliers comparing to MSE. MAE measures the average absolute difference between actual and predicted values. By looking at MAEs we should choose with the lowest MAE if we want a metric that gives equal weight to all errors. Other than MSE and MAE metrics we have R2 score. It represents the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, where 1 indicates a perfect fit. In the case of choosing best model with R2 score, we should consider the model with the highest R2 score. But at the same time, we should be careful with overfitting issue. You can see the best method that I chose under the next subtitle after applying ordinary-least-squared method.

Results and Explanation

The Best Model with f value 10:
LinearRegression() with
Mean-Squared-Error: 443.1639611822952
Mean-Absolute-Error: 15.64879809384971
R2 Score: 0.34033074304637934

Linear Equation for the Best Model:

*y = 179.33326271186442 + (-4.378438224533927 * X_0) + (8.994608244325995 * X_1) + (10.251934509971468 * X_2) + (-7.6056272879295195 * X_3) + (-5.024886200394534 * X_4) + (4.017933015095547 * X_5) + (-0.4472457286448114 * X_6) + (-0.8208998761859805 * X_7) + (-2.197764718164857 * X_8) + (3.792463024046885 * X_9)*

When I look at the results of the metrics of the three models, the model with f value 10 has lowest MSE and highest R2 score. Since we want lower MSE, MAE and higher R2 score it can be a good choice among three models. When look at the MAE for the model with f value 10, it is not the lowest among three model but actually it is not a bad value when comparing all of them. Also we have better results with MSE and R2 score other than the other models. So the best model with f value 10 is the best model that I chose.

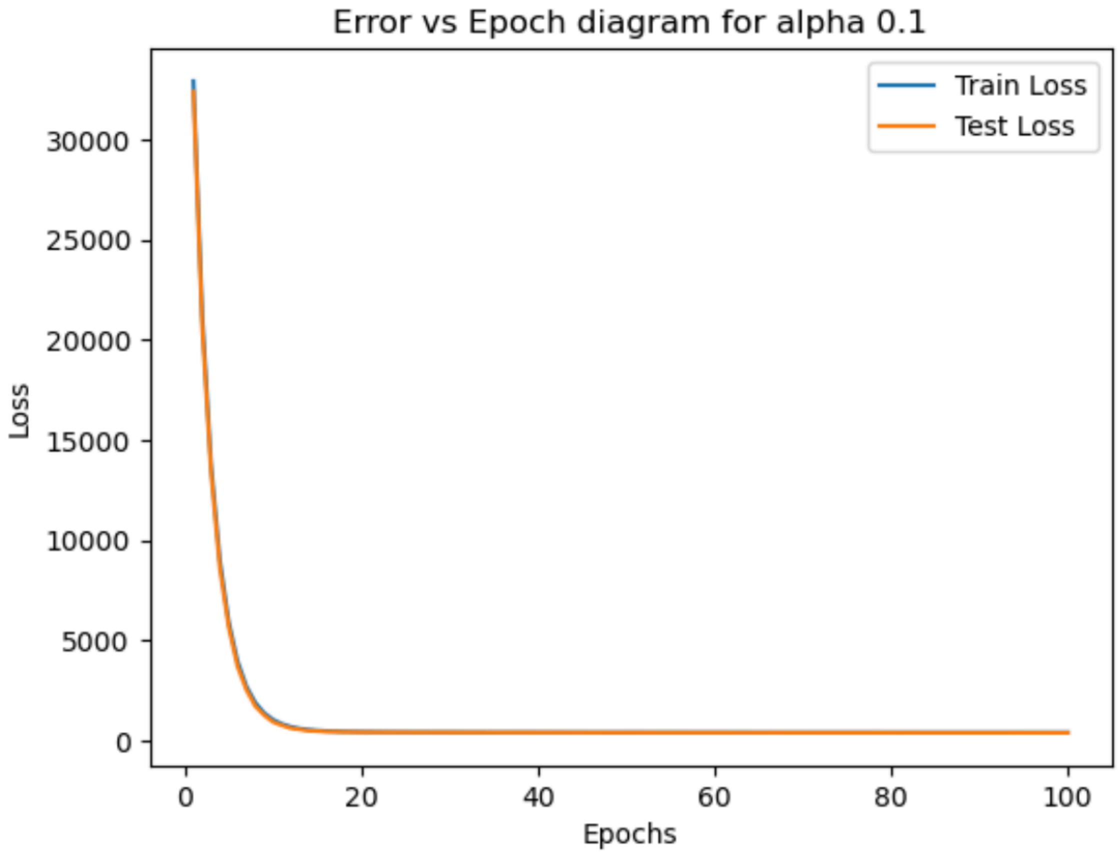
4. Gradient Descent

Implementation

```
def gradient_descent(X_train, X_test, y_train, y_test, alpha, numOfIterations):

    # 'Weights' correspond to coefficients for the features in the input data.
    # 'Bias' corresponds to intercept in the linear equation
    weights = np.zeros(X_train.shape[1])
    bias = 0

    # 'Epochs' correspond to the number of times the gradient descent
    # algorithm will iterate through the entire dataset
    epochs = numOfIterations

    # Declaring numpy arrays in order to store the error
    # losses calculated for training and test sets here
    trainLosses = []
    testLosses = []

    # Each time iterate through the entire dataset
    for ep in range(epochs):

        # Calculating predictions and loss for the training set
        yTrainPred = np.dot(X_train, weights) + bias
        trainLoss = np.mean(np.square(yTrainPred - y_train))
        trainLosses = np.append(trainLosses ,trainLoss)

        # Calculating predictions and loss for the test set
        yTestPred = np.dot(X_test, weights) + bias
        testLoss = np.mean(np.square(yTestPred - y_test))
        testLosses = np.append(testLosses,testLoss)

        # Updating weights and bias using gradient descent in
        # the opposite direction of the gradients in order
        # to minimize the loss
        weights = weights - alpha * (2/len(X_train)) * np.dot(X_train.T, (yTrainPred - y_train))
        bias =  bias - alpha * (2/len(X_train)) * np.sum(yTrainPred - y_train)

    return weights, bias, trainLosses, testLosses

# Train models with different learning rates
learningRates = [0.1, 0.5, 0.01]

for alpha in learningRates:
    weights, bias, trainLosses, testLosses = gradient_descent(X_trainScaled, X_testScaled, y_train, y_test,
                                                              alpha, 100)
    print(f"Results for learning rate {alpha}:")
    print(f"Weights: {weights}, Bias: {bias}")
    # Plot the epoch-loss diagram
    plt.plot(range(1, len(trainLosses)+1), trainLosses, label='Train Loss')
    plt.plot(range(1, len(testLosses)+1), testLosses, label='Test Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title(f'Error vs Epoch diagram for alpha {alpha}')
    plt.legend()
    plt.show()
    print(f"----Losses for train data for the model with alpha val: {alpha}----")
    print(trainLosses)
    print("\n")
    print(f"----Losses for test data for the model with alpha val: {alpha}----")
    print(testLosses)
    print("-----------------------------------------------------\n")
```

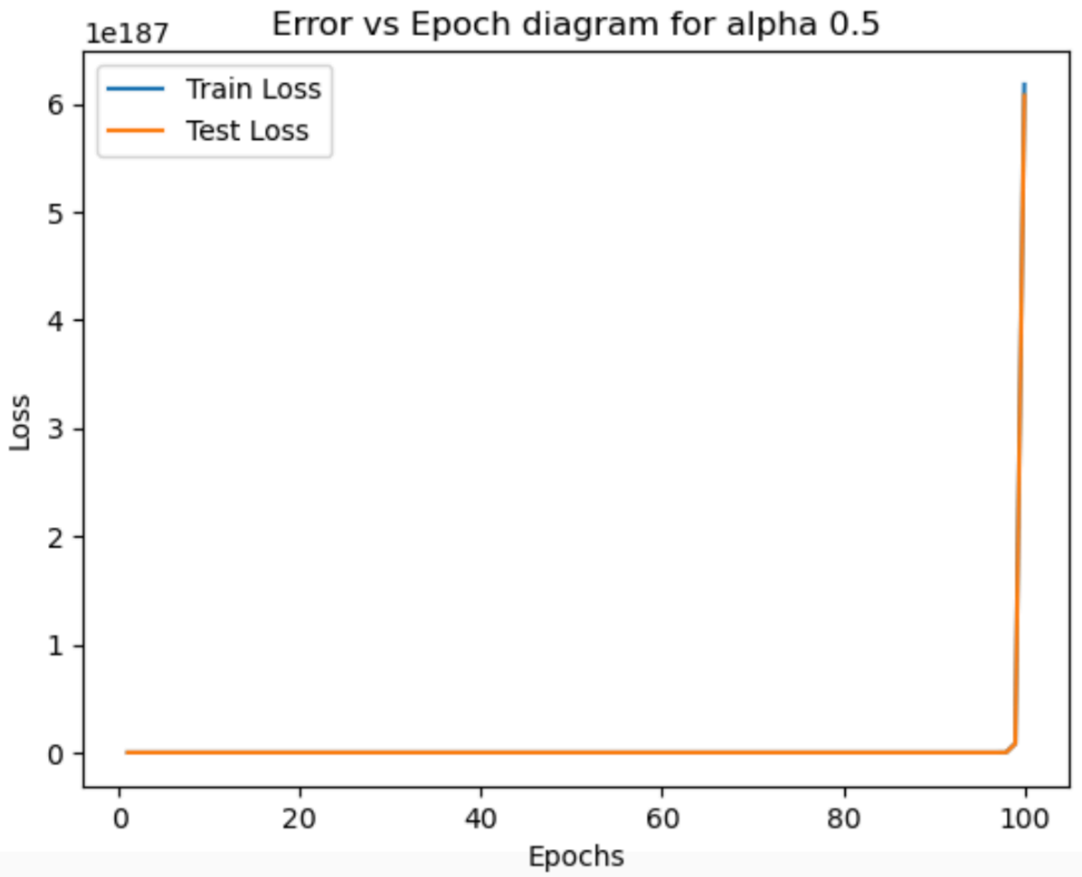
Model Evaluation

```
Results for learning rate 0.1:
Weights: [-0.92531662  3.09442635  8.35931988  0.64072013 -1.27041053 -0.30584311
 -0.32105436 -1.24775457 -0.85493468 -0.97927522  0.53846628  6.64534416
 -1.97006177  2.55388191  0.32916207 -3.00216434  3.19750232 -5.20617308
 -4.20315507  2.67502082 -4.54455691 -1.58242223  5.68186561 -2.65197347
  2.0254707   3.0561517   2.57411658 -0.06229792 -3.26228576 -8.85048084
 -0.22830118], Bias: 179.3332626753336
```



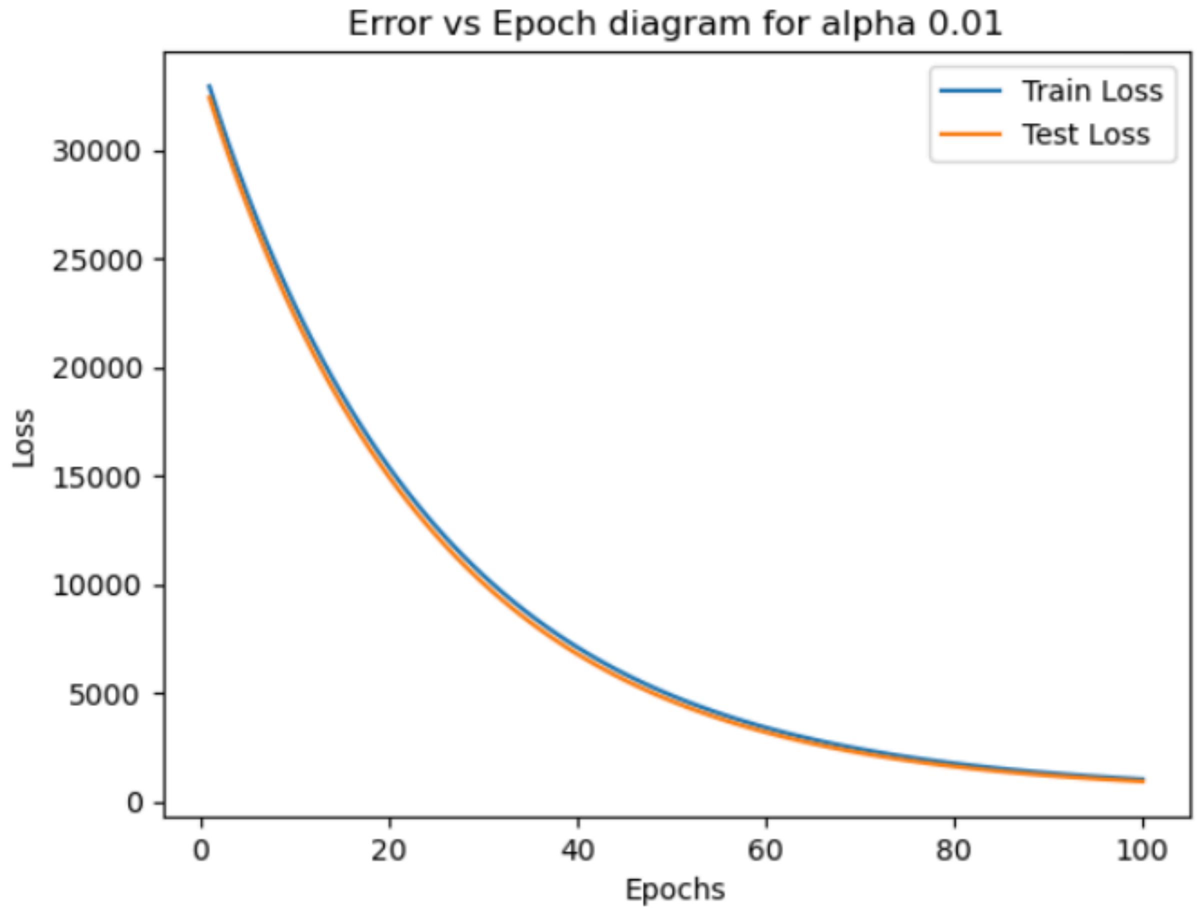
```
----Losses for train data for the model with alpha val: 0.1----
[32920.56489407 21234.79277401 13764.27188872  8984.01177293
  5923.16770606  3962.01516558  2704.53932501  1897.52206472
  1378.98575924  1045.2850814    830.08166426   690.90336955
   600.54913619   541.59119701   502.85785034   477.18295149
   459.96570787   448.24888533   440.12871977   434.37709433
   430.1995659    427.08034557   424.6829447    422.78646287
   421.24470207   419.95990372   418.86585685   417.91701499
   417.08146817   416.33639063   415.66508025   415.05502294
   414.49661852   413.98233458   413.50613833   413.06310996
   412.64917518   412.26091659   411.89543802   411.55026446
   411.22326681   410.91260368   410.61667571   410.33408879
   410.06362411   409.8042133    409.55491768   409.31491068
   409.08346298   408.85992963   408.6437392    408.4343842
   408.23141303   408.0344228    407.84305327   407.65698149
   407.47591719   407.29959877   407.12778975   406.96027579
   406.79686199   406.6373706    406.48163898   406.32951786
   406.18086982   406.03556789   405.89349444   405.75454012
   405.61860297   405.48558764   405.35540469   405.22796999
   405.10320421   404.98103231   404.86138317   404.74418922
   404.6293861    404.51691239   404.40670939   404.29872086
   404.19289282   404.08917343   403.98751277   403.88786274
   403.79017694   403.69441052   403.60052014   403.50846383
   403.41820092   403.32969199   403.24289877   403.15778412
   403.07431192   402.99244709   402.91215547   402.83340384
   402.75615984   402.68039194   402.60606944   402.53316238]

----Losses for test data for the model with alpha val: 0.1----
[32401.88453782 20656.25079738 13391.26263086  8598.52586271
  5670.13409717  3704.84933614  2539.58048971  1730.11449335
  1274.7240024    938.24832606   765.54209492   623.19864707
   561.24500207   499.01887152   479.29752639   450.47162791
   446.083954     431.4477072    432.09609297   423.69971833
   425.61400161   420.12875747   422.04504359   418.04620976
   419.62352585   416.4768153    417.68769028   415.09166568
   415.99669098   413.79385433   414.46435529   412.56244585
   413.05815757   411.39678161   411.76254507   410.29855347
   410.56662149   409.26732649   409.460609     408.30035105
   408.43519721   407.39340327   407.48166319   406.54161702
   406.59207515   405.74006367   405.75940512   404.98408841
   404.97754117   404.26947115   404.24123123   403.59247722
   403.54599103   402.94984589   402.8879995    402.33874814
   402.26399596   401.7567326    401.67118656   401.20167008
   401.10716293   400.67170219   400.56983368   400.16519619
   400.05736783   399.68070678   399.56814876   399.21694429
   399.10073704   398.77274872   398.65384067   398.34706871
   398.22629117   397.93894454   397.81702454   397.54749446
   397.42506597   397.17190371   397.04951751   396.81141562
   396.68954811   396.46532441   396.3443855    396.13296932
   396.01330956   395.81372975   395.69564672   395.50702125
   395.39076539   395.21229218   395.09807201   394.92902083
   394.8170077    394.656713     394.54704534   394.39489988
   394.28768712   394.14313623   394.03846223   393.90099874]
```

```
Results for learning rate 0.5:
Weights: [ 2.60644398e+93  2.19856606e+93  6.13395671e+92  6.08415788e+93
  2.16445955e+93 -5.79491680e+93  1.19916945e+93 -3.26268814e+92
 -9.49942622e+92 -1.46442328e+93  2.79880262e+92  2.46132430e+93
 -3.59961421e+93 -3.01519696e+93  3.37001344e+93  4.51302734e+93
 -3.45396704e+93  5.52850764e+93  5.91273898e+93 -4.32691273e+93
  6.34976602e+93  6.63613473e+93  6.10154142e+93 -6.18290553e+93
 -6.20648346e+93  2.45284622e+93 -2.68093589e+93  3.22018795e+93
  1.01270930e+92  3.05933231e+93 -6.33129499e+92], Bias: 2.6682362639989975e+77
```



```
----Losses for train data for the model with alpha val: 0.5----
[3.29205649e+004 1.16429103e+004 8.36009003e+005 6.26290834e+007
 4.69552586e+009 3.52060538e+011 2.63970445e+013 1.97922120e+015
 1.48399898e+017 1.11268677e+019 8.34280819e+020 6.25534973e+022
 4.69019537e+024 3.51665910e+026 2.63675397e+028 1.97701037e+030
 1.48234156e+032 1.11144409e+034 8.33349079e+035 6.24836366e+037
 4.68495729e+039 3.51273165e+041 2.63380921e+043 1.97480241e+045
 1.48068606e+047 1.11020282e+049 8.32418383e+050 6.24138539e+052
 4.67972506e+054 3.50880858e+056 2.63086773e+058 1.97259692e+060
 1.47903240e+062 1.10896292e+064 8.31488725e+065 6.23441492e+067
 4.67449867e+069 3.50488989e+071 2.62792954e+073 1.97039390e+075
 1.47738060e+077 1.10772442e+079 8.30560106e+080 6.22745223e+082
 4.66927812e+084 3.50097557e+086 2.62499462e+088 1.96819333e+090
 1.47573064e+092 1.10648730e+094 8.29632525e+095 6.22049732e+097
 4.66406340e+099 3.49706563e+101 2.62206299e+103 1.96599522e+105
 1.47408252e+107 1.10525155e+109 8.28705979e+110 6.21355018e+112
 4.65885450e+114 3.49316006e+116 2.61913463e+118 1.96379957e+120
 1.47243624e+122 1.10401719e+124 8.27780467e+125 6.20661079e+127
 4.65365142e+129 3.48925884e+131 2.61620954e+133 1.96160637e+135
 1.47079180e+137 1.10278421e+139 8.26855990e+140 6.19967916e+142
 4.64845416e+144 3.48536199e+146 2.61328772e+148 1.95941562e+150
 1.46914920e+152 1.10155260e+154 8.25932545e+155 6.19275526e+157
 4.64326269e+159 3.48146948e+161 2.61036916e+163 1.95722732e+165
 1.46750843e+167 1.10032237e+169 8.25010131e+170 6.18583910e+172
 4.63807703e+174 3.47758132e+176 2.60745386e+178 1.95504145e+180
 1.46586950e+182 1.09909352e+184 8.24088747e+185 6.17893066e+187]


----Losses for test data for the model with alpha val: 0.5----
[3.24018845e+004 1.13419999e+004 8.21452994e+005 6.13834578e+007
 4.60811077e+009 3.45843909e+011 2.59448760e+013 1.94583635e+015
 1.45915438e+017 1.09412526e+019 8.20387303e+020 6.15126174e+022
 4.61218113e+024 3.45817543e+026 2.59290728e+028 1.94413591e+030
 1.45769311e+032 1.09296311e+034 8.19492294e+035 6.14446715e+037
 4.60705685e+039 3.45432275e+041 2.59001484e+043 1.94196586e+045
 1.45606556e+047 1.09174262e+049 8.18577126e+050 6.13760511e+052
 4.60191169e+054 3.45046493e+056 2.58712228e+058 1.93979705e+060
 1.45443940e+062 1.09052335e+064 8.17662927e+065 6.13075054e+067
 4.59677221e+069 3.44661140e+071 2.58423294e+073 1.93763065e+075
 1.45281506e+077 1.08930544e+079 8.16749749e+080 6.12390362e+082
 4.59163846e+084 3.44276218e+086 2.58134683e+088 1.93546668e+090
 1.45119254e+092 1.08808889e+094 8.15837590e+095 6.11706436e+097
 4.58651045e+099 3.43891725e+101 2.57846395e+103 1.93330512e+105
 1.44957182e+107 1.08687369e+109 8.14926451e+110 6.11023273e+112
 4.58138817e+114 3.43507661e+116 2.57558428e+118 1.93114598e+120
 1.44795292e+122 1.08565985e+124 8.14016329e+125 6.10340873e+127
 4.57627161e+129 3.43124027e+131 2.57270783e+133 1.92898924e+135
 1.44633582e+137 1.08444737e+139 8.13107223e+140 6.09659235e+142
 4.57116076e+144 3.42740821e+146 2.56983459e+148 1.92683492e+150
 1.44472054e+152 1.08323625e+154 8.12199133e+155 6.08978359e+157
 4.56605562e+159 3.42358043e+161 2.56696456e+163 1.92468300e+165
 1.44310705e+167 1.08202647e+169 8.11292057e+170 6.08298242e+172
 4.56095618e+174 3.41975692e+176 2.56409774e+178 1.92253349e+180
 1.44149537e+182 1.08081805e+184 8.10385994e+185 6.07618886e+187]
-----------------------------------------------------
```

```
Results for learning rate 0.01:
Weights: [ 0.05267679  0.96669533  7.58546084 -0.85728114  0.11199243  0.71898088
 -0.0140828  -0.82632621 -0.74459755 -0.76179112 -0.18090837 -0.26527586
 -1.64260107  2.75569092 -0.21300241 -2.44603547  3.79154552 -3.54573718
 -1.64426964  2.48563643 -1.00910168 -0.14862673  1.87535418  0.46335149
  1.5820054   0.95509913  1.11911036 -0.54268668 -3.08032326 -2.20577531
 -0.61503452], Bias: 155.55016505385987
```



```
----Losses for train data for the model with alpha val: 0.01----
[32920.56489407 31584.80725505 30318.22586744 29112.45910501
 27961.43378387 26860.591853   25806.38593359 24795.94991882
 23826.88357843 22897.11142012 22004.78993107 21148.24635198
 20325.93801488 19536.42510189 18778.35217384 18050.43543921
 17351.45378966 16680.24231591 16035.6874655  15416.72329503
 14822.32845964 14251.52370592 13703.36971511 13176.96519571
 12671.44515922 12185.97933456 11719.77069185 11272.05405523
 10842.09479106 10429.1875617  10032.65513781  9651.84726408
  9286.13957414  8934.9325516   8597.6505345   8273.74076074
  7962.67245269  7663.93593901  7377.04181202  7101.52011913
  6836.91958686  6582.80687606  6338.76586709  6104.39697359
  5879.31648391  5663.15592875  5455.56147431  5256.19333957
  5064.72523696  4880.84383533  4704.24824449  4534.64952023
  4371.77018914  4215.34379242  4065.11444792  3920.83642956
  3782.27376362  3649.19984108  3521.39704545  3398.65639535
  3280.77720147  3167.56673704  3058.83992155  2954.419017
  2854.13333626  2757.81896303  2665.31848299  2576.48072554
  2491.16051597  2409.21843736  2330.52060197  2254.93843183
  2182.34844788  2112.63206767  2045.67541092  1981.36911303
  1919.60814574  1860.29164519  1803.32274656  1748.60842548
  1696.05934562  1645.58971236  1597.11713233  1550.56247843
  1505.84976031  1462.9059999   1421.66111197  1382.04778936
  1344.00139281  1307.45984516  1272.36352973  1238.65519274
  1206.2798496   1175.18469496  1145.31901627  1116.6341108
  1089.08320594  1062.62138269  1037.20550217  1012.79413501]

----Losses for test data for the model with alpha val: 0.01----
[32401.88453782 31057.81388555 29786.37996763 28578.53506528
 27427.64369862 26328.68984072 25277.75492894 24271.67319667
 23307.80326954 22383.87611765 21497.89325328 20648.05807281
 19832.7291268  19050.38795055 18299.61660552 17579.08173247
 16887.5229994  16223.7445394  15586.60844173 14975.02966901
 14387.97197808 13824.44455819 13283.49919092 12764.22779677
 12265.76027478 11787.26256885 11327.93491349 10887.01022517
 10463.75261371 10057.45599546  9667.44279363  9293.06271506
  8933.69159461  8588.73030039  8257.60369408  7939.75964174
  7634.66807119  7341.82007264  7060.72703963  6790.91984784
  6531.94806949  6283.37922126  6044.79804415  5815.80581336
  5596.01967692  5385.07202159  5182.60986472  4988.29427098
  4801.79979272  4622.81393305  4451.03663051  4286.17976448
  4127.96668053  3976.13173466  3830.41985586  3690.58612605
  3556.39537688  3427.62180249  3304.04858774  3185.46755121
  3071.67880238  2962.49041243  2857.71809812  2757.18491822
  2660.72098192  2568.16316891  2479.35486046  2394.14568123
  2312.39125129  2233.95294793  2158.69767698  2086.49765315
  2017.23018909  1950.77749277  1887.02647297  1825.8685524
  1767.19948825  1710.9191999   1656.93160338  1605.14445244
  1555.46918594  1507.82078122  1462.11761337  1418.28131999
  1376.23667142  1335.91144601  1297.23631042  1260.14470463
  1224.57273149  1190.45905074  1157.74477711  1126.37338262
  1096.29060258  1067.44434549  1039.78460639  1013.26338375
   987.83459957   963.45402275   940.07919541   917.66936219]
----------------------------------------------------
```

Results and Explanation

When we look at the diagram and loss values of the model with an alpha value of 0.1, we see that the loss value decreases very quickly in the first iterations. At the same time, by comparing the minimum loss values achieved by all 3 models, the model with the alpha value of 0.1 provides us the lowest loss values at the end of the epochs. Therefore for the best model I chose the model with alpha value of 0.1.

5. KNN - Regression

Implementation

```python
from sklearn.neighbors import KNeighborsRegressor

def knn_regression(X_train, X_test, y_train, y_test, k):
    # Applying KNN Regressor
    model = KNeighborsRegressor(n_neighbors=k)
    # Fit the model
    model.fit(X_train, y_train)
    # Predict
    yPred = model.predict(X_test)
    # Calculate MSE
    mse = mean_squared_error(y_test, yPred)

    return model, mse

# K values
kValues = [3, 5, 10]
results = {}
for k in kValues:
    model, mse = knn_regression(X_trainScaled, X_testScaled, y_train, y_test, k)
    # Using dictionary for results
    results[k] = {
        'model': model,
        'mse': mse
    }
    print(f"Results for K={k}:")
    print(f"MSE: {mse}")
    print("    =", round(mse))
    print("----")

# Identifying the best model (The best model is with lowest MSE)
bestModelKey = min(results, key=lambda k: results[k]['mse'])
bestModel = results[bestModelKey]['model']

print("---- RESULT ----")
# Printing the best model with k value
print("The Best Model with k value:")
print(f"{bestModel} with mse:", results[bestModelKey]['mse'])
```

Model Evaluation

```
Results for K=3:
MSE: 495.06725490196055
   = 495
----
Results for K=5:
MSE: 478.1488806722688
   = 478
----
Results for K=10:
MSE: 476.60784285714266
   = 477
----
```

Results and Explanation

```
---- RESULT ----
The Best Model with k value:
KNeighborsRegressor(n_neighbors=10) with mse: 476.60784285714266
```

As a result, when we compare all models, we should choose the model with the lowest mean-squared-error value as the best model. Because, lower mean-squared-error indicates better model performance. For this reason, I chose the best model among these 3 models as the model with k value 10 that has the lowest MSE value.